**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

дисциплина «Введение в нейронные сети»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

«Контролируемое обучение нейронных сетей»

Вариант № 6

Выполнил:

Студент: группы ИТП-31

Гошко В.В.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2022

**Задание:** разработать программное обеспечение для решения задачи бинарной классификации с учителем с использованием простейших нейронных сетей.

Архитектуру нейронной сети выбрать самостоятельно. На входном слое не должно быть менее 25 нейронов и не более 36. Количество скрытых слоёв не должно быть менее 1 и более 3-х. Изображения использовать бинаризованные. В качестве первичного алгоритма обучения использовать алгоритм, указанный в таблице, согласно варианта. Дополнить разработанное программное обеспечение методом обратного распространения ошибки (backpropagation). Сравнить с помощью графиков качество и скорость обучения одной и той же сети различными методами.

Вариант задания на рисунке 1.

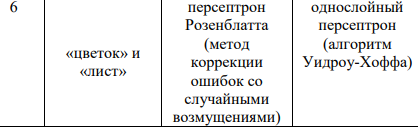


Рисунок 1 – вариант задания

**Ход работы:**

Первым делом был создан файл, содержащий обучающую и тестовую выборки. Бинаризованные цифры на рисунке 2.

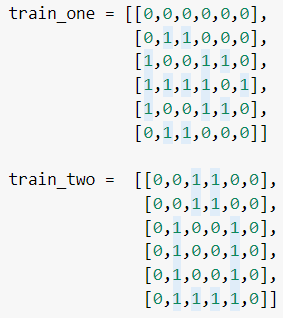


Рисунок 2 – фрагмент бинаризованных цифр для обучения сети

Результат выполнения и проверки на рисунке 2.



Рисунок 2 – результат выполнения и проверки нейронной сети

**Вывод:** было разработано программное обеспечение для решения задачи бинарной классификации с учителем с использованием простейших нейронных сетей и нейронной сети обратного распространения ошибки.

**ПРИЛОЖЕНИЕ А**

Листинг

import numpy as np

def get\_train\_neurons():

    train\_zero =[[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    train\_one = [[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    train\_two =  [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    train\_three = [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    train\_four = [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    train\_five =[[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    train\_six =  [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    train\_seven= [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    train\_eight= [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    train\_nine = [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    train\_numbers = [train\_zero, train\_one, train\_two, train\_three, train\_four,

                    train\_five, train\_six, train\_seven, train\_eight, train\_nine]

    converted\_numbers\_to\_neurons = list()

    for item in train\_numbers:

        converted\_numbers\_to\_neurons.append([i for sublist in item for i in sublist])

    return converted\_numbers\_to\_neurons

def get\_test\_neurons():

    test\_zero = [[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    test\_one =  [[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    test\_two =  [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    test\_three = [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    test\_four =  [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    test\_five = [[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    test\_six =  [[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    test\_seven = [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    test\_eight = [[0,0,1,1,1,0],

                  [0,0,1,0,1,0],

                  [0,0,1,1,1,0],

                  [0,1,0,1,0,0],

                  [0,0,1,1,0,0],

                  [0,0,0,1,0,0]]

    test\_nine = [[0,0,1,0,0,0],

                 [0,1,1,0,0,0],

                 [0,1,1,1,0,0],

                 [1,1,1,1,1,0],

                 [1,1,1,1,1,0],

                 [0,0,1,0,0,0]]

    test\_numbers = [test\_zero, test\_one, test\_two, test\_three, test\_four,

                    test\_five, test\_six, test\_seven, test\_eight, test\_nine]

    converted\_numbers\_to\_neurons = list()

    for item in test\_numbers:

        converted\_numbers\_to\_neurons.append([i for sublist in item for i in sublist])

    return converted\_numbers\_to\_neurons

#Реализация при помощи сети Уидроу-Хоффа

#На входном слое 36 нейронов

#На скрытом слое 10 нейронов

class NeuralNet:

    def \_\_init\_\_(self):

        self.weights\_x\_h = np.random.normal(size=(36, 10))

        self.weights\_h\_o = np.random.normal(size=(10, 10))

        self.sigmoid\_mapper = np.vectorize(self.sigmoid) # Функция, котороя позволяет применить ко всему вектору функцию sigmoid

    def sigmoid(self, x): # Функция нормализации сигмоид

        return 1 / (1 + np.exp(-x))

    def predict(self, input): # Предсказание (input == x)

        h\_summator = np.dot(input, self.weights\_x\_h) # Суммирует произведения входящих нейронов на веса, ведущие к нейрону скрытого слоя

        h = self.sigmoid\_mapper(h\_summator) # Вектор значений нейронов скрытого слоя после нормализации

        o\_summator = np.dot(h, self.weights\_h\_o)

        o = self.sigmoid\_mapper(o\_summator) # Вектор значений выходных нейронов

        print(o)

        result = "\nПроверка\n"

        for i in range(10):

            if o[i] > 0.05:

                result += f"Это листик\n"

            else:

                result += f"Это цветок\n"

        return result + "\n"

    def train(self, X\_train, y\_true, epochs): # Обучение сети методом обратного распространения ошибки (input == x, expected == y)

        mini\_chec = True

        i = 0

        while mini\_chec :

            err\_f = 0

            for input, expected in zip(X\_train, y\_true):

                input = np.transpose(input) # Превращаем векторы в столбцы-векторы

                expected = np.transpose(expected)

                # Проходим вперед и ищем вектор o (1x10) - предсказания

                h\_summator = np.dot(input, self.weights\_x\_h)

                h = self.sigmoid\_mapper(h\_summator)

                o\_summator = np.dot(h, self.weights\_h\_o)

                o = self.sigmoid\_mapper(o\_summator)

                # Ищем ошибку и распространяем её назад, находя изменения весов

                # Ищем ошибку со скрытого слоя на выходной и дельту w от o

                error\_o = expected - o

                err\_f += error\_o

                grad\_o = o \* (1 - o)

                delta\_w\_o = error\_o \* grad\_o

                # Ищем ошибку с входного слоя на скрытый и дельту w от h

                error\_h = np.dot(delta\_w\_o, self.weights\_h\_o.T)

                grad\_h = h \* (1 - h)

                delta\_w\_h = error\_h \* grad\_h

                # Изменяем веса

                self.weights\_x\_h += np.dot(input.reshape(36, 1), delta\_w\_h.reshape(1, 10))

                self.weights\_h\_o += np.dot(h.reshape(10, 1), delta\_w\_o.reshape(1, 10))

            for err in range(len(error\_o)):

                if abs(error\_o[err]) < 0.01:

                    mini\_chec = False

            if i % 10 == 0:

                print(f"Эпоха {i} {np.average(err\_f)}")

            i = i + 1

#Получение данных

#import train\_test\_data as data

def get\_data():

    X\_train = get\_train\_neurons()

    X\_test = get\_test\_neurons()

    y = []

    for i in range(10):

        y.append(list())

        for j in range(10):

            if j == i:

                y[i].append(1)

            else:

                y[i].append(0)

    X\_train = X\_train + X\_test

    y = y + y

    return np.array(X\_train), np.array(X\_test), np.array(y)

X\_train, X\_test, y\_true = get\_data()

nn = NeuralNet()

nn.train(X\_train, y\_true, 1000)

#Проверка на тестовой выборке

print(nn.predict(X\_test[0]))